MODIS
SCIENCE DATA SUPPORT TEAM
PRESENTATION

April 10, 1992

AGENDA

ACTION ITEMS

01/17/92 [Tom Goff]: Have a polished version (with peer review) of the file dump routine ready for the MODIS Science Team Meeting. (The routine has been reviewed by the SDST software review committee, and their changes have been incorporated.) **STATUS: Open. Due date 04/01/92.**

02/21/92 [Lloyd Carpenter and Team]: Identify a list of risks associated with porting Team Members' algorithms to the PGS. Prepare these for discussion at the Science Team Meeting. (An updated version of the list is included in the handout.) **STATUS: Open. Due date 04/01/92.**

03/20/92 [Lloyd Carpenter]: Gather the MODIS Data Product Attributes information, and write a cover letter to Team Members for updating the information, and discussion at the Team Meeting. (Completed) **STATUS: Open. Due date 03/27/92.**

04/03/92 [Team]: Provide comments and questions (to Jim Ormsby and Al Fleig) on geolocation and registration (for discussion with the land team). **STATUS: Open. Due date 04/10/92.**

04/03/92 [Team]: Provide comments to Jim Ormsby on the MODIS-N SDST Glossary of Definitions (handed out at the 02/28/92 meeting). **STATUS: Open. Due date 04/10/92.**

MODIS Airborne Simulator (MAS) Level-1 Processing System

Status and Data Availability

April 13, 1992

Liam E. Gumley

MODIS Science Data Support Team

Research and Data Systems Corporation

Introduction

The MODIS Airborne Simulator (MAS) Level-1 processing system was designed and implemented by the MODIS Science Data Support Team at GSFC during the last half of 1991, and processed the first MAS flight data in November 1991. The purpose of the processing system is to ingest MAS Level-0 aircraft sensor, engineering and navigation data, and produce calibrated, geolocated radiances in a portable format. This document briefly describes

- the MAS instrument specifications,
- the calibration and geolocation procedures for the MAS,
- the processing system operation and output products,
- current data processing status and availability.

MAS instrument description

Platform	NASA ER-2 aircraft
Ground speed	206 meters per second (nominal)
Altitude	18 kilometers (nominal)
Total field of view	85.92 degrees
Swath width	33.52 kilometers (at 18 kilometers altitude)
Instantaneous field of view	2.5 milliradians
Pixel spatial resolution	45 meters (at 18 kilometers altitude)
Pixels per scan line	716
Scan rate	6.25 scan lines per second
Data channels	12
Bits per channel	8 (configured to have 4 channels @ 10 bits, 7 channels @ 8 bits)
Data rate	232.2 Megabytes/hour (Ames Level-0 intermediate format)
Spectral channels	50 (subset of 12 selected for flight)
Visible calibration	Integrating sphere on the ground - none onboard
Infrared calibration	Two temperature controlled black bodies on board

MAS Visible/Near-Infrared Calibration

The MAS does not have an onboard visible/near-infrared (VIS/NIR) calibration capability. An integrating sphere is used on the ground before and after flight missions to generate calibration data for the VIS/NIR channels. A linear calibration of the form

$$\text{Radiance} = \text{Slope} \times \text{Count} + \text{Intercept}$$

is used to compute VIS/NIR radiances. It is currently assumed that this calibration is not time dependent and does not change during flight. However recent instrument characterization at Ames Research Center has revealed that VIS/NIR channel sensitivity may decrease as the MAS instrument cools during ascent to cruise altitude. Efforts are underway to characterize

this effect, and an updated calibration method which corrects for the sensitivity changes with temperature will be implemented. More details on the MAS VIS/NIR calibration scheme are given in Jedlovec et. al. (1).

MAS Infrared Calibration

Calibration data for the MAS infrared (IR) channels is obtained during flight from two temperature controlled blackbody sources. These blackbodies are maintained at temperatures of approximately 238 K (-35 C) and 273 K (0 C) respectively. These are viewed during every mirror scan. The calibration algorithm for these channels involves

- computing equivalent Planck radiances at the blackbody temperatures,
- computing the calibration slope and intercept by the relationships

$$\text{Slope} = \frac{\text{Radiance(BB2)} - \text{Radiance(BB1)}}{\text{Count(BB2)} - \text{Count(BB1)}}$$

$$\text{Intercept} = \frac{\text{Radiance(BB1)} \times \text{Count(BB2)} - \text{Radiance(BB2)} \times \text{Count(BB1)}}{\text{Count(BB2)} - \text{Count(BB1)}}$$

where

Radiance(BB1), Radiance(BB2) are the equivalent Planck radiances for the cool (238 K) and warm (273 K) blackbodies respectively,

Count(BB1), Count(BB2) are the digital instrument counts for the cool (238 K) and warm (273 K) blackbodies respectively,

- computing the equivalent sensor radiance for each pixel by the relationship

$$\text{Radiance} = \text{Slope} \times \text{Count} + \text{Intercept}.$$

This procedure is performed for every IR channel on every scanline. No averaging of MAS blackbody data is done. More details on the MAS IR calibration scheme are given in Jedlovec et. al. (1).

MAS Geolocation

Geolocation data for the MAS is recorded continuously during flight by the ER-2 Inertial Navigation System (INS). The important parameters are time, latitude, longitude, heading and altitude. The INS updates these values every 5 seconds.

The geolocation algorithm is only applied to portions of a flight where the aircraft flew a straight and level line. Straight line flight tracks are identified by manual inspection of the change in aircraft heading with time. Linear regressions for aircraft latitude, longitude, heading and altitude versus time are computed for the straight line flight tracks.

To geolocate a given MAS straight line flight track, the MAS start time and scanline number at the beginning of the flight track are determined. These are used as a reference for the rest of the flight track, since MAS times are truncated to whole seconds. The scanline number and scan rate are used to determine the time elapsed to subsequent scanlines in the flight track. Once the time for a given scanline is computed, the linear regression relationships are used to compute aircraft latitude, longitude, heading and altitude at that time. Latitudes and longitudes are then computed for every 10th pixel on that scanline (pixels 1, 10, 20, 30, ..., 690, 700, 710, 716). Solar zenith and azimuth angles, and aircraft scan and azimuth angles are also computed for every 10th pixel. Every scanline in a straight line flight track is geolocated in this way. Scanlines which are not included in straight line flight tracks have no geolocation data computed. However it should be noted that the INS data is still available during these sections. More details on the MAS geolocation scheme are given in Jedlovec et. al. (1).

MAS Level-1 Data Processing System

The MAS Level-1 Data Processing System was designed at GSFC on DEC VAX/VMS and Silicon Graphics Iris systems in FORTRAN77, and currently runs on either system. The present processing configuration uses a VAX system as a MAS Level-0 data staging area, and a Silicon Graphics Iris as a processing platform.

The current sequence of processing steps is:

(1) Read MAS Level-0 tapes onto VAX,
(2) Check Level-0 data for anomalies or problems,
(3) Determine flight track time limits from INS data,
(4) Compute navigation regressions from time limit data,
(5) Compute calibration data,
(6) Create calibrated, geolocated data set for each flight line,
(6) Move Level-1 data to FTP site,
(7) Writing Level-1 data summary for FTP site.

The data is currently distributed by File Transfer Protocol (FTP) over Internet. A facility for distributing data on Exabyte 8mm tapes will exist in the near future.

MAS Level-1B data format

During the design of the MAS Level-1 processing system, it was decided that many of the problems associated with dataset portability and usability could be solved by creating the MAS

Level-1B datasets in a standard, portable format. After some investigation, the Network Common Data Form (NetCDF) interface (2) was selected as the means for reading and writing the MAS Level-1B data files. NetCDF was developed by the Unidata Program Center at the University Corporation for Atmospheric Research (UCAR). The purpose of NetCDF is to allow the user to create, access, and share scientific data in a form that is self-describing and network-transparent. "Self-describing" means that a file includes information defining the data it contains. "Network-transparent" means that a file is represented in a form that can be accessed by computers with different ways of storing integers, characters, and floating-point numbers.

NetCDF data files are accessed through C or FORTRAN interfaces which are provided by UCAR. The interfaces have been successfully tested by UCAR on the following platforms:

- Sun 3, SPARCstation (SunOS)
- DEC VAX (VMS, Ultrix)
- DECstation (Ultrix)
- IBM RISC System 6000 (AIX)
- Cray YMP (UNICOS)
- IBM PS/2 (MSDOS, OS/2)

The NetCDF interface library source code is available from UCAR by anonymous FTP. Instructions on how to obtain the software are appended to this document (Appendix 1).

MAS Level-1B data contents

The product generated by the MAS Level-1 processing system is defined to be calibrated, geolocated radiances. When designing the MAS Level-1B dataset, a prime concern was that all Level-0 engineering and ancillary input data be maintained in the output data set.

Each individual Level-1B dataset contains calibrated, geolocated radiances for all MAS channels for one straight line flight track, or flight line. Each Level-1B flight line is contained within one NetCDF file. A description of a Level-1B flight line file is appended to this document (Appendix 2).

MAS Level-1B data availability

The MAS Level-1B data is currently distributed at GSFC by anonymous FTP via Internet. Details on how this may be done are appended to this document (Appendix 3). Since many of the typical MAS flight line files are tens of megabytes in size, FTP transfers to remote sites can be slow. For this reason, a facility to distribute MAS data on Exabyte 8mm tapes is under development. Plans are also underway to develop an on-line catalog system for the MAS data.

MAS data processing status

The current inventory of MAS Level-0 data at GSFC includes 1 test flight, 1 ferry flight, 11 science flights (FIRE) and 3 calibration runs (FIRE). The next major MAS mission will be the ASTEX deployment in June 1992. The following table lists the MAS data processed so far at GSFC, as of 9 April 1992.

Flight Date	Area covered during flight	Level-0 data received	Processing completed	INS offset fixed
10/31/91	Ames test flight CA/NV	yes	3/3 tracks	yes
11/12/91	Ferry flight CA to TX	yes (subset)	1/1 tracks	no
11/14/91	Coffeyville KS	yes	16/16 tracks	no
11/18/91	Coffeyville KS	yes	14/14 tracks	yes
11/21/91	Coffeyville KS	yes		
11/22/91	Coffeyville KS	yes		
11/24/91	Gulf coast TX/LA	yes		
11/25/91	Coffeyville KS	yes		
11/26/91	Coffeyville KS	yes		
12/03/91	Gulf coast TX/LA	yes		
12/04/91	Gulf coast TX/LA	yes		
12/05/91	Coffeyville KS	yes	29/29 tracks	no
12/07/91	Coffeyville KS	yes		
11/16/91	Ground visible calibration	yes	10481 scanlines (no navigation)	
11/20/91	Ground visible calibration	yes	6078 scanlines (no navigation)	
11/23/91	Ground visible calibration	yes	10281 scanlines (no navigation)	

Appendices 4, 5 and 6 show some of the summary data generated for the MAS FIRE flight on 18 November 1991.

References

(1) Improved Capabilities of the Multispectral Atmospheric Mapping Sensor (MAMS), January 1989, G.J. Jedlovec et. al., NASA TM 100352, Marshall Space Flight Center.

(2) NetCDF Users Guide, An Interface for Data Access, Version 2.0, October 1991, Unidata Program Center, University Corporation for Atmospheric Research.

Appendix 1

Instructions for obtaining the netCDF library source code and documentation

The following sequence of commands and command output demonstrates how to retrieve the netCDF library source code and documentation from UCAR via anonymous FTP. The computer system used was a Silicon Graphics Iris with the Irix 3.3.2 operating system.

```
% ftp unidata.ucar.edu
Connected to unidata.ucar.edu.
220 groucho FTP server (SunOS 4.1) ready.
Name (unidata.ucar.edu:gumley): anonymous
331 Guest login ok, send ident as password.
Password:
230 Guest login ok, access restrictions apply.
ftp> cd pub
250 CWD command successful.
ftp> binary
200 Type set to I.
ftp> get netcdf-2.02.tar.Z
local: netcdf-2.02.tar.Z remote: netcdf-2.02.tar.Z
200 PORT command successful.
150 Binary data connection for netcdf-2.02.tar.Z (128.183.252.9,1288) (830751
bytes).
226 Binary Transfer complete.
830751 bytes received in 100.72 seconds (8.05 Kbytes/s)
ftp> quit
221 Goodbye.
% uncompress netcdf-2.02.tar.Z
% tar xvof netcdf-2.02.tar
```

This netCDF version (2.02) was downloaded on 11 March 1992.
Installation questions or problems should be addressed to Russ Rew at UCAR (russ@unidata.ucar.edu).

Appendix 2

Structure of a MAS Level-1B NetCDF flight track file

The following information summarizes the variables in a MAS flight track file. All MAS flight track files have this same format.

dimensions:
This section defines the dimensions of the variables contained in the data set.

Time = UNLIMITED ; // (2601 currently)
This dimension is the 'length' dimension of the data set. It is extendable in the sense that extra data can be added at the end of the data set. Physically this dimension corresponds to elapsed time, however it also corresponds to the incrementing of the scan line counter.

NumberOfChannels = 12 ;
This is the number of channels on the MAS.

NumberOfPixels = 716 ;
This is the number of pixels across a MAS scan.

HeaderLength = 1840 ;
This is the length of the ASCII data set header. This is composed of blocks of 80 characters, which contain no carriage return, line feed or similar record separators. Currently the header contains one line of descriptive text that is entered by the person who processed the data, followed by 22 lines which describe the MAS instrument configuration as used in processing. This text is self-documenting.

AnchorIndexSize = 73 ;
This is the number of geolocation anchor points per scan line.
Each MAS scan line has geolocation data for every 10th pixel (to save space). The geolocation data is defined for pixel numbers
1, 10, 20, 30, 40,.....680, 690, 700, 710 and 716
which makes a total of 73 geolocation anchor points.
It should be noted that pixels 1 to 358 are on the starboard (right) side of the aircraft, while pixels 359 to 716 are on the port (left) side of the aircraft.

variables:
This section defines the type and size of the variables in the output data set.
The type definitions are from the C language, however the equivalent FORTRAN types are
char = CHARACTER,
short = INTEGER*2,
long = INTEGER*4,
float = REAL*4.

char DataSetHeader(HeaderLength) ;
ASCII header text.
This indicates which spectral bands were selected, which channels had 8 or 10 bit data, which channels were calibrated using the MAS blackbodies (IR channels), and slopes/intercepts for the visible/near-IR channels.

short AnchorPtIndex(AnchorIndexSize) ;
Pixel numbers for which geolocation information is defined.

short DataFrameStatus(Time) ;
(Level-0 MAS engineering data)
Zero indicates good data.

long ScanLineCounter(Time) ;
(Level-0 MAS engineering data)
Scan line count, increments by 1 for every scan line.

long ThumbWheelSwitches(Time) ;
(Level-0 MAS engineering data)
Data system thumbwheel switch settings.

short ScanRate(Time) ;
(Level-0 MAS engineering data)
Scan rate in scans per second (x 10, nearest integer).

long GMTime(Time) ;
(Level-0 MAS engineering data)
Greenwich Mean Time (HHMMSSS).

short S-BendIndicator(Time) ;
(Level-0 MAS engineering data)
S-bend indicator : 0=no S-bend, 1=S-bend.

short AircraftRollCount(Time) ;
(Level-0 MAS engineering data)
Aircraft roll count (signed integer, positive is right), 0.03 degrees per count.

long Year&DayOfYear(Time) ;
Year, month, day (YYYYMMDD).

short BlkBdy1Temperature(Time, NumberOfChannels) ;
(Level-0 MAS engineering data)
Black Body 1 (cold) thermal reference temperature
(degrees C x 100)

short BlkBdy2Temperature(Time, NumberOfChannels) ;
(Level-0 MAS engineering data)
Black Body 2 (hot) thermal reference temperature
(degrees C x 100)

short AmplifierGain(Time, NumberOfChannels) ;
(Level-0 MAS engineering data)

Instrument Gain (x 1000)

short BlkBdy1Counts(Time, NumberOfChannels) ;
(Level-0 MAS engineering data)
Black Body 1 (cold) radiance count

short BlkBdy2Counts(Time, NumberOfChannels) ;
(Level-0 MAS engineering data)
Black Body 2 (hot) radiance count

float CalibrationSlope(Time, NumberOfChannels) ;
Count to radiance calibration slope.
Radiance = (count*slope + intercept)/gain (visible/near-IR channels),
Radiance = count*slope + intercept (IR channels).
Radiance units are:
milliWatts per square centimeter per steradian per micron
for visible/near-IR channels (calibrated by integrating sphere),
milliWatts per square centimeter per steradian per wavenumber
for IR channels (calibrated using MAS blackbodies).

float CalibrationIntercept(Time, NumberOfChannels) ;
Count to radiance calibration intercept.
Units are the same as the calibration slope.

float PixelLatitude(Time, AnchorIndexSize) ;
Latitudes for pixels at geolocation anchor points.
Latitude ranges from -90 degrees at the South Pole to
+90 degrees at the North Pole.

float PixelLongitude(Time, AnchorIndexSize) ;
Longitudes for pixels at geolocation anchor points.
Longitude is zero at the Greenwich Meridian, and ranges
from -180 degrees (West) to +180 degrees (East).

float SensorZenithAngle(Time, AnchorIndexSize) ;
MAS sensor zenith angle for pixels at geolocation anchor points.
Defined as the zenith angle (degrees) of a vector from
the sensor to the pixel (nadir sensor zenith angle = 0).

float SensorAzimuthAngle(Time, AnchorIndexSize) ;
MAS sensor azimuth angle for pixels at geolocation anchor points.
Defined as the azimuth angle (degrees) clockwise from
North of a vector from the pixel to the sensor.

float SolarZenithAngle(Time, AnchorIndexSize) ;
Solar zenith angle for pixels at geolocation anchor points.
Defined as the zenith angle (degrees) of a vector from
the pixel to the Sun.

float SolarAzimuthAngle(Time, AnchorIndexSize) ;
Solar azimuth angle for pixels at geolocation anchor points.
Defined as the azimuth angle (degrees) clockwise from
North of a vector from the pixel to the Sun.

float AircraftLatitude(Time) ;
Aircraft subpoint latitude (degrees, derived from INS data).

float AircraftLongitude(Time) ;
Aircraft subpoint longitude (degrees, derived from INS data).

float AircraftHeading(Time) ;
Aircraft heading (degrees, derived from INS data).

float AircraftAltitude(Time) ;
Aircraft altitude (meters, derived from INS data).

float AircraftPitch(Time) ;
Aircraft pitch angle (degrees, derived from INS data).

short CalibratedData(Time, NumberOfChannels, NumberOfPixels) ;
Calibrated MAS radiances for all channels and all pixels
(x 100, nearest integer).
Radiance units are:
milliWatts per square centimeter per steradian per micron
for visible/near-IR channels (calibrated by integrating sphere),
milliWatts per square meter per steradian per wavenumber
for IR channels (calibrated using MAS blackbodies).

Appendix 3

Accessing the MAS anonymous FTP site

An anonymous FTP account has been set up to enable users of MAS data to become familiar with the structure of MAS Level-1B datasets, and to provide a limited distribution mechanism for MAS Level-1B data. Questions should be directed to:

Liam E. Gumley (RDC 301-982-3748 gumley@ltp.gsfc.nasa.gov), or
Thomas E. Goff (RDC 301-982-3704 teg@ltp.gsfc.nasa.gov).

Connecting to the anonymous FTP site

The files in the account have been set up specifically for Silicon Graphics Iris users. It is therefore advisable to use your own Iris as a base for retrieving files. However, you can still retrieve data using any computer with FTP. To connect to the host, type

```
ftp ltpiris2.gsfc.nasa.gov
```

(or if that doesn't work)

```
ftp 128.183.252.9
```

For a username, enter

```
anonymous
```

and for a password, enter your Internet ID, e.g.

```
hoges@barbie.gsfc.nasa.gov
```

You should then change to the MAS directory by typing

```
cd pub/MAS
```

Getting files from the anonymous FTP area

Files in the area are either ASCII or BINARY. In order to transfer them correctly to your own machine, you must tell your FTP program what type of file you are going to fetch. For example, to get the BINARY file shrimp.dat, you would type

```
binary
get shrimp.dat
```

Similarly, to get the ASCII file downunder.doc, you would type

```
ascii
get downunder.doc
```

Please ensure you use the correct transfer mode to avoid possible file corruption. Note that MAS Level-1B data files (with .cdf extension) are always in BINARY form.

Directories and files in the anonymous FTP area

The first file you will see is 00readme.doc - you are now reading it.

There are several directories present.

Directory util contains source code, executable programs, and netCDF libraries for use on the Iris. The files in this directory are

Size	Name	Description
58172	fdump	(BIN) file dump program (Iris only)
18737	fdump.c	(ASC) file dump C source code
94376	libnetcdf.a	(BIN) netCDF v2.02 library (Iris only)
424176	masdump	(BIN) MAS file dump program (Iris only)
9251	masdump.f	(ASC) MAS file dump FORTRAN source code
151388	ncdump	(BIN) netCDF file dump program (Iris only)
1532	netcdf.doc	(ASC) how to get netCDF source via FTP
10162	netcdf.h	(ASC) netCDF v2.02 C header file
5050	netcdf.inc	(ASC) netCDF v2.02 FORTRAN include file
132444	netcdf2pci	(BIN) netCDF to PCI program (Iris only)
5922	netcdf2pci10.c	(ASC) netCDF to PCI C source code
134156	subset	(BIN) netCDF image subset program (Iris only)
11659	subset31.c	(ASC) netCDF image subset C source

Several other directories will be present, each of which contains MAS Level-1B flight lines. The syntax of the directory names will be something like

12nov91

which represents the day, month and year of the flight in question. Each directory will contain files from just one flight. Each file will have a name like

12nov91-01.cdf

where the '-01' represents the flight line number. These files are typically large (many megabytes) and are BINARY in form. They may take a significant period of time to transfer with FTP. Included in each of these directories will also be a file named something like

12nov91.doc

which is an ASCII text file containing a brief summary of the flight. Also, a file named something like

12nov91.ins

will be present, which is a BINARY file containing the ER-2 Inertial

Navigation System (INS) data for the flight. These files have 150 bytes per line, and are ASCII text (but use BINARY transfer for FTP).

Appendix 4

MAS Flight Summary

Date : 18-NOV-1991
Takeoff : 1920Z
Landing : 0120Z
Airfield : Ellington AFB, Houston TX
Aircraft : ER-2 709
Destination : Coffeyville KS
Purpose : FIRE deployment (cirrus cloud study)
Scanlines : 68973
Channels : 12 (2 - 8 @ 8 bits, 9 - 12 @ 10 bits, 1 was bit bucket)
IFOV : 2.5 milliradians
MAS clock : Internal
INS clock : Internal
Clock offset: INS - MAS = 65.06 seconds CORRECTED

Total of 14 straight line flight tracks

Number	Start Time	End Time	(Times in decimal hours)
01	20.551	20.586	
02	20.614	20.860	
03	20.942	21.018	
04	21.028	21.112	
05	21.168	21.340	
06	21.404	21.582	
07	21.639	21.806	
08	21.860	22.043	
09	22.061	22.236	
10	22.275	22.475	
11	22.522	22.650	
12	22.704	22.892	
13	22.954	23.124	
14	23.185	23.351	

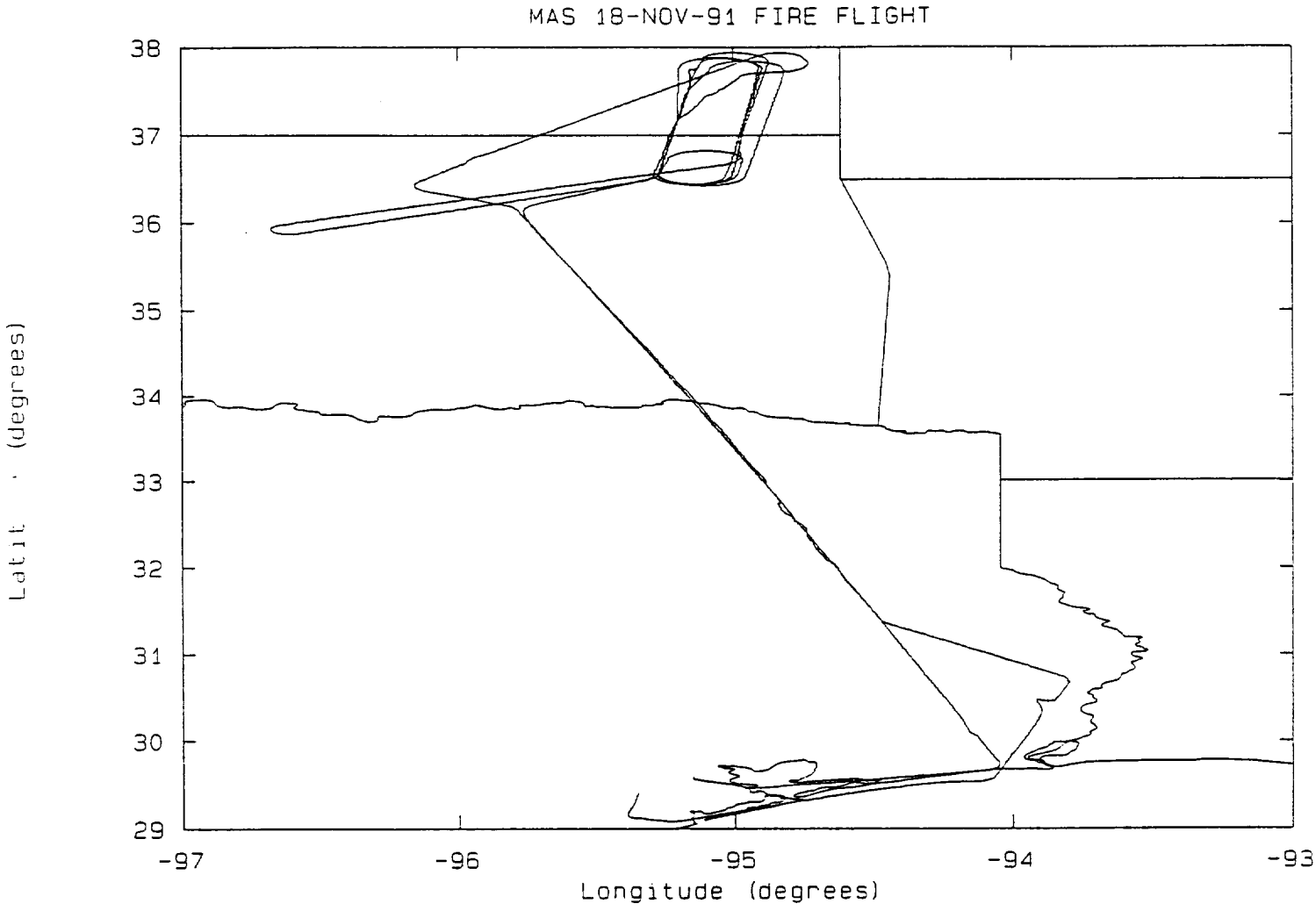
Appendix 5

MAS Level-1B flight line summary

Date	18-NOV-1991
Start time	221639.00 hours
End time	222818.00 hours
Nominal heading	62 degrees
Nominal altitude	19903 meters
Number of scan lines	4351
Start, end scan line numbers	68691 73046
Nominal BB1,BB2 temperatures	-37.43 C, -0.72 C
First valid navigated record	1
Nadir start lat,lon	35.964, -96.697 degrees
Top left lat,lon	35.819, -96.597 degrees
Top left solar zen,azm	79.865, 237.111 degrees
Last valid navigated record	4351
Nadir end lat,lon	36.585, -95.243 degrees
Bottom right lat,lon	36.737, -95.333 degrees
Bottom right solar zen,azm	83.214, 239.730 degrees

Appendix 6

ER-2 flight track for MAS 18 November 1991 FIRE mission



DRAFT

MODIS Science Data Support Team (SDST)

Coding Recommendations for the MODIS Science Team

DRAFT

Table of Contents

1 Introduction

This is the first draft version of the coding recommendations for the MODIS Science Team. The current schedule calls for the next update in January 1993. The ß version is scheduled for July 1993. Versions 1 and 2 are scheduled for July of 1994 and 1995 respectively. Corrections and suggested changes are welcome at any time. Commented examples will be included in the next update to these recommendations.

1.1 Objective

The objective of these recommendations is to facilitate the porting, integration, testing, documentation, and maintenance of code for the generation of MODIS science data products on an operational basis. These recommendations are designed to assist the MODIS Science Team Members in preparing their code for this process.

1.2 Scope

The intent of this document is not to tell you how to write your algorithms, but to provide recommendations for writing code meant to be ported to the EOS Product Generation System (PGS) by the SDST.

1.3 Languages/Operating System

All MODIS science data processing code to be ported to the MODIS Team Leader Computing Facility (TLCF) and integrated into the control shell with other code will be written in either Standard Fortran or Standard C programming language, without extensions. The PGS and the MODIS TLCF are planned to conform to the Portable Operating System Interface for UNIX (POSIX) and the Government Open Systems Interconnections Profile (GOSIP).

1.4 Code Checkers

We are looking into the use of code checkers, such as QA Fortran, Fortran Lint and FTN Check as a first step in the porting process. We recommend the use of code checkers during code development to identify problems at an early stage.

1.5 Software Deliverables

Software packages delivered to the SDST for porting to the PGS should contain:

- source code, installation instructions and "make" file
- operating instructions, and other documentation,
- test input and output data sets which test all paths in the code
- test drivers, especially when the software is not "stand-alone".

2 Readability/Documentation

Efficient porting, integration, testing, documentation, and maintenance of code depends heavily on being able to read and understand the code (including the documentation). The code must be written and documented so as to be read, understood, and used by another knowledgeable person or group of people in a different environment. Readability, or understandability, is the most important criterion for successful implementation of MODIS code.

2.1 Documentation

Proper software documentation covers a wide range of topics including theory, programmer's guide, user's guide, etc. While all available code documentation should be provided to the SDST, the present focus is on internal documentation of source code.

Comments are interspersed throughout a module, whereas documentation occurs at the beginning of a module. Documentation is specific, and it should be revised to reflect changes. Completeness and readability, not brevity, are the main considerations. For each code module the documentation should contain, at a minimum, a standard prologue consisting of the following:

- a one-line header with the module name and a short description,
- a version line with the revision code and date,
- the author, sponsor and institution originating the module,
- the purpose of the module and instructions on its use,
- a list of called functions and subroutines,
- definition, description and units of the input, output and internal variables,
- method used, if applicable,
- references and credits,
- notes and warnings (conscious design limitations).

Input variables should be given thorough and complete definitions and descriptions. Units, type, dimensions, special cases, upper and lower limits, usage examples, default values, and interrelations with other input variables should all be provided.

2.2 Declarations

Declarations and data statements should be placed just before the first executable statement of the module.

2.3 Variable Names

All variable names should be meaningful to the knowledgeable reader. The use of short or cryptic variable names should be avoided wherever possible. A variable name within a computer program should have only one meaning within the context of that program, and should be used for only one purpose. The use of Fortran or C language keywords as variable names should be avoided.

2.4 Structure

The code should be adequately and logically blocked, commented and indented to clearly show the structure and logical flow. Comments should be uniformly set-off from the code. All transfers of control and destinations should be clearly annotated. A path must be defined for every possible outcome of a logical decision. The level of nesting should be kept to a minimum. Statement labels (if used) should occur in a clear and natural sequence. There should be only one statement per line.

2.5 Modularity/Cohesiveness

Complex programs can generally be separated in a logical way into modules (subprograms) each of which does a single task. When each module is cohesive, and the coupling between modules is loose, the code becomes more understandable, more testable, more maintainable, and more easily documented. The aim is to achieve a level of modularity which keeps the individual modules cohesive and comprehensible while avoiding the clutter of too many relatively trivial modules. Avoid combining several functions together arbitrarily. Modules should generally be limited in length to 1 or 2 pages.

3 Portability

Portability is of primary concern in the case of MODIS science code which is developed in one environment and implemented for production processing in a different environment. The degree of concern increases with the degree of difference between the two environments.

3.1 Language/Operating System

The best guideline to achieve portability is to adhere to standard FORTRAN or C, and avoid the use of system-dependent features.

Many portability problems are best avoided by specifying a small set of primitive operations for accessing the environment. Operating system dependencies are then confined to a small number of procedures and functions, so the code can be moved to a UNIX system where the primitives can be implemented.

3.2 Data Portability

Many portability problems can arise if care is not taken to design portable (machine-independent) data sets. It is strongly recommended that the use of portable data formats be investigated. Examples are the Network Common Data Form (NetCDF) and the Hierarchical Data Format (HDF). References for these formats are provided.

If a standard portable data format is not used, then special care must be taken in the design of data sets which are to be ported to different environments. Floating point data, for example, is often highly machine specific. Data sets which contain integer or ASCII data present the fewest portability problems as long as sufficient ancillary information is provided regarding word lengths, byte ordering, and so on. Data sets with mixed data types such as floating point, integer and character data present the greatest portability problems.

Each data set should have a header explaining the contents, origin, format, etc. of the data set.

4 Testability

Test data and test results should be included with the delivery of the code and documentation to the SDST. The porting and integration process is not complete until the code has been successfully tested on the TLCF. These tests must give results which are consistent with testing done prior to delivery to the SDST.

4.1 Module Testing

Each module should be tested independently of the total program. The test should include exercising every logical branch in each module, checking for possible failure, checking for reasonableness of results, comparison of limiting cases with analytic solutions, comparison with published results where possible. Tests should also include cases of invalid or implausible input variables, no matter how unlikely it is that the module will be used incorrectly. In the code, each error message should be kept together with its associated error check.

4.2 Test Drivers

A comprehensive test driver will explore all of the branches of a program and compare results with "correct answers" which are included in the driver code. A good test driver will also "failure-test" the model by pushing it into regimes where trouble is expected. Good test drivers, developed with good coding practices, are very helpful in testing, porting, integrating, and maintaining code. Test drivers are recommended for MODIS science code and they should be supplied to the SDST.

5 Maintainability

Considering the duration of the MODIS mission, all code will be subject to maintenance (changes, and updating). The maintenance process will be greatly simplified if the recommendations addressed elsewhere in this document are applied in the design and development of code.

Appendix A: Software Code Evaluation Criteria

This list of software code evaluation criteria is provided as a suggested checklist for code review. Some of the items apply to more than one category.

Readability/Documentation

Does each module have the standard prologue?

Are definitions, descriptions and units given for all variables?

Are declarations and data statements placed after the prologue and before the first executable statement?

Are variable names meaningful?

Do variable names avoid the use of language keywords?

Has structured programming been utilized?

Is the code logically and adequately commented, blocked and indented to show structure?

Are comments uniformly set-off from code?

Are all transfers of control and destinations annotated?

Is a path defined for every possible outcome of a logical decision?

Is the level of nesting kept to a minimum?

If statement labels are used, do they occur in a clear and natural sequence?

Is there only one statement per line?

Is the program divided into cohesive and comprehensible modules of reasonable size?

Portability

Is the software written in Standard FORTRAN or Standard C without extensions?

Are operating system dependencies limited to a small set of primitive operations?

Are mixed binary arrays of fixed and floating point numbers avoided?

Testability

Are test data and test results provided with the code?

Are limit checks performed to ensure that variable contents are within the expected range of values?

Are input defaults explicitly tested?

Are errors and associated error messages kept together?

Other Criteria

Calling subroutines

Do arguments in call statements not contain arithmetic or logical expressions?

Does each module contain a single entry point and a single exit point?

Are shared variables communicated as arguments whenever practical to ensure program modularity? (Minimize the use of COMMON and EQUIVALENCE.)

Error handling and prevention

Are flagged/missing data values correctly excluded from calculations?

Is the code designed to handle errors and failures gracefully?

Are possibilities for infinite loops avoided?

Are stack overflow problems avoided?

Variables, constants & parameters

Are constants defined? Are counters, variables and parameters initialized?

Are local variables within modules declared as static (type) where appropriate?

Are loop index parameters and array subscripts expressed only as integer constants or integer variables?

References

Warren J. Wiscombe, "Principles of Numerical Modeling with Examples from Atmospheric Radiation" 1989, unpublished.

NCSA HDF Calling Interfaces and Utilities, Version 3.1, July 1990, University of Illinois at Urbana-Champaign

NetCDF User's Guide, An Interface for Data Access, Version 2.0, October 1991, Unidata Program Center, University Corporation for Atmospheric Research

Upper Atmosphere Research Satellite Software Assurance Recommendations Document, Version 3.0, NASA/GSFC, July 1989

Things that will Increase the Risks in Porting Code

General

Delays in the software delivery will increase the risk arising from schedule pressure on the SDST.

Readability

Increased risk of misinterpretation will result from:

- poorly written and poorly documented code,
- the use of cryptic or meaningless variable names,
- omitting the standard prologue,
- having Team Members and/or their programmers unavailable to answer questions.

Portability

Increased portability risk will result from:

- the use of non-standard FORTRAN or non-standard C,
- the use of machine specific language extensions or "tricks",
- the use of machine dependent internal and external data sets,
- development of code on a type of machine to which the SDST does not have access.

Testability

Increased testability risk will result from:

- missing or inadequate test drivers, test data and/or test results,
- inadequately modularized code,

Operability

Increased operability risk will result from:

- requirements for human intervention,
- excessive use of COMMON and EQUIVALENCE,
- missing or inadequate error messages.